The Neural MoveMap Heuristic in Chess

Levente Kocsis, Jos W.H.M. Uiterwijk, Eric Postma, and Jaap van den Herik

Department of Computer Science, Institute for Knowledge and Agent Technology,
Universiteit Maastricht, The Netherlands
{l.kocsis,uiterwijk,postma,herik}@cs.unimaas.nl

Abstract. The efficiency of alpha-beta search algorithms heavily depends on the order in which the moves are examined. This paper investigates a new move-ordering heuristic in chess, namely the Neural MoveMap (NMM) heuristic. The heuristic uses a neural network to estimate the likelihood of a move being the best in a certain position. The moves considered more likely to be the best are examined first. We develop an enhanced approach to apply the NMM heuristic during the search, by using a weighted combination of the neural-network scores and the history-heuristic scores. Moreover, we analyse the influence of existing game databases and opening theory on the design of the training patterns. The NMM heuristic is tested for middle-game chess positions by the program Crafty. The experimental results indicate that the NMM heuristic outperforms the existing move ordering, especially when a weighted-combination approach is chosen.

1 Introduction

Most game-playing programs for two-person zero-sum games use the alpha-beta algorithm. The efficiency of alpha-beta is closely related to the size of the expanded search tree and depends mainly on the order in which the moves are considered. Inspecting good moves first increases the likelihood of cut-offs that decrease the size of the search tree.

For move ordering, the existing programs frequently rely on information gained from previous phases of the search. Major examples of such information support are transposition tables [1] and the history heuristic [2]. Additionally, the original move-ordering techniques which are independent of the search process are still valid. In chess, for example, checking moves, captures and promotions are considered before 'silent' moves.

The search-independent move-ordering techniques are usually designed based on information provided by a domain expert. Recently two attempts were proposed to obtain a better move ordering by using learning methods. The chessmaps heuristic [3] employs a neural network to learn the relation between the control of the squares and the influence of the move. In a similar way, we trained a neural network to estimate the likelihood of a move being the best in a certain position [4]. We termed this technique the Neural MoveMap (NMM) heuristic. There are two main differences between the chessmaps heuristic and the NMM heuristic. First, in the chessmaps heuristic the neural network is trained to evaluate classes of moves, while in the NMM heuristic the neural network is tuned

J. Schaeffer et al. (Eds.): CG 2002, LNCS 2883, pp. 154–170, 2003.

to distinguish between individual moves. Second, in the chessmaps heuristic the target vector represents the influence of the move on the squares of the board, while in the NMM heuristic the target information for a neural network is the likelihood of the move being the best in a certain position.

The training information of the NMM heuristic shares common ideas with the comparison paradigm [5, 6] developed for learning evaluation functions. In the original version designed by Tesauro, a neural network is trained to compare two board positions. The positions are encoded in the input layer, and the output unit represents which of the two is better. The comparison paradigm was also employed to evaluate moves in Go [7]. Accordingly, a neural network was trained to rate the expert moves higher than random moves. The technique is not efficient for move ordering in a search-intensive game program due to speed limitations.

In [4], we presented the results of the NMM heuristic in the game of Lines of Action (LOA). The promising results were confirmed when the heuristic was tested in MIA [8], one of the top LOA programs. The reduction of the search tree was more than 20 percent, with an overhead of less than 8 percent [9]. Despite this success, it was unclear how the NMM heuristic performs when it is included in tournament programs for more complex and, especially, more studied games. In this article we investigate the performance of the NMM heuristic in chess by inserting the move ordering in the strong tournament program Crafty [10]. In LOA, we inserted the NMM heuristic in the search by replacing the move ordering of the history heuristic by that of the neural network. In this article we improve upon this approach. Chess, compared to LOA, has two extra features that can influence the training of the neural network: the existence of game databases and the advanced opening theory. In the experiments we analyse the choices resulting from these two features too.

The article is organized as follows. Section 2 describes the NMM heuristic. The experimental set-up for evaluating the efficiency of the NMM heuristic is described in Sect. 3. The experimental results are given in Sect. 4. Finally, Sect. 5 presents our conclusions.

2 The Neural MoveMap Heuristic

For the NMM heuristic, a neural network is trained to estimate the likelihood of a move being the best in a certain position. During the search, the moves considered more likely to be the best are examined first. The essence of the heuristic is rather straightforward. However, the details of the heuristic are complex since they are crucial for the heuristic to be effective, i.e., to be fast and to result in a small search tree. The details include: the architecture of the neural network (Sect. 2.1), the construction of the training data (Sect. 2.2) and the way the neural network is used for move ordering during the search (Sect. 2.3).

2.1 The Architecture of the Neural Network

For the game of LOA, we analysed several architectures for the neural network [4]. We found that the best architecture encodes the board position in the input

units of the neural network and uses one output unit for each possible move of the game. A move is identified by its origin and destination square (i.e., the current location and the new location of the piece to move). The activation value of an output unit corresponding to a move represents the score of that move. The other analysed architectures include a more compact representation of the moves, either in the input or in the output of the neural network.

The same move encoding can be used for chess. In the case of LOA, we assigned one input unit to each square of the board, with +1 for a black piece, −1 for a white piece and 0 for an empty square. In chess there are more piece types, and consequently we assign 6 input units (one for every piece type) to each square. An additional unit is used to specify the side to move. Consequently, the resulting network has 385 (6×64+1) input units and 4096 (64×64) output units.

Although the network is very large, the move scores can be computed quickly, since we have to propagate only the activation for the pieces actually on the board, and to compute only the scores for the legal moves. To increase the speed further, in the case of chess we removed the hidden layer, present in the network used for LOA. For LOA, we already noticed that in this architecture the hidden layer is not increasing significantly the performance [4]. This way, the resulting move ordering requires just a little extra computation during the search, namely a summation over the pieces on the board.

2.2 The Construction of the Training Data

The neural network described in Sect. 2.1 performs a linear projection, and any learning algorithm should thus be reasonably fast. Consequently, we can use any of the existent learning algorithms for neural networks without influencing significantly the training. However, the choice of the training data is an important issue. A training instance consists of a board position, the legal moves in the position and the move which is the best. From these three components, determining the legal moves by an algorithm poses no problem. The choices on the other two components are more difficult. There are two questions to be answered: "Where are the best moves coming from?" and "Which positions should be included in the training phase?"

In each training instance, one of the legal moves is labeled as the best. The labeling can have two sources: (1) the move played in the game (as specified in the database), (2) the move suggested by a game program (the one that will use the neural network for move ordering). If we choose the second source, the neural network might be able to incorporate the program's bias (e.g., preference to play with bishops). The second source has, however, the disadvantage of the extra computation needed for analysing all the positions, since, as observed for LOA, a large number of positions have to be used to obtain good results. In the experimental section, we examine both sources.

Chess games are usually divided into three phases: opening, middle game and endgame. In each of these phases different strategies are used. We focus on middle-game positions. There the original opening usually has a substantial influence on the strategies employed by chess players. In the experiments,

we investigate whether it is better to have opening-specific neural networks, or whether it is sufficient if only one network is used for all middle-game positions. The training positions originate from game databases either clustered by opening or collected in a mixed form.

2.3 Using the Neural Network during the Search

When the neural network is used during the search the moves are ordered according to the network's estimation of how likely a certain move is the best. The move ordering has to be placed in the context of the move orderings already existent in the game program. In the following, we describe three approaches to include the neural network in the search: the pure neural-network approach, the straightforward-combination approach, and the weighted-combination approach. These approaches deal only with the moves that would be ordered by the history heuristic.

Pure Neural-Network Approach: The first approach to use the neural network during the search is by replacing in every node of the search tree the move ordering of the history heuristic by that of the neural network. In the following, we refer to this approach as the pure neural-network approach. This approach was tested for LOA [4].

Straightforward-Combination Approach: The move ordering of the neural network and that of the history heuristic are not necessarily mutually exclusive; they can be combined. Hence, the second approach investigated in this article, the straightforward-combination, is as follows. We first consider the move rated best by the neural network followed by the moves as ordered by the history heuristic (of course, excluding the move picked by the neural network).

Weighted-Combination Approach: The history-heuristic score is built up from information collected during the search process. Therefore, it has a dynamic nature, but it has no specific connection with the position under investigation. In contrast, the scores suggested by the neural network are specific to the position; they are static, and do not gain from the information obtained in the current search process. In principle, a combination of the two scores can benefit both from information obtained in the search, and from information obtained off-line by analysing a large number of positions. Such a combination can be more suitable for the position in which the moves have to be ordered.

One way to combine the history-heuristic score and the neural-network score is by adding them. Since the two scores are not in the same range at least one of them has to be scaled.

The distribution of the neural-network scores is specific to a certain neural network and does not depend on the search depth or opening line of a position[1]. A sample distribution is given in Fig. 1, left. The neural-network scores

[1] Although the independence of the opening line might seem counter-intuitive, this is what we observed experimentally.

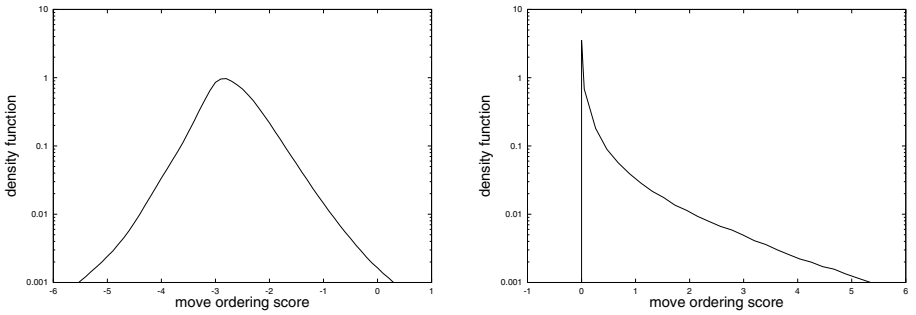

Fig. 1. The distribution of scores for the neural network (left) and the history heuristic (right).

are approximately normally distributed, and centered around a negative value, since most moves are not considered to be best. The history-heuristic scores are known to become larger as the search progresses. One way to normalize the history-heuristic scores is to divide them by the total number of history updates. After this normalization, the history-heuristic scores in the different parts of the search tree are approximately in the same range. However, they are still in a range different from the neural-network scores. Therefore, we divide the normalized history-heuristic scores by a coefficient that we term the history weight. The resulting distribution, using the value 500 for the history weight (see below and Sect. 4.2), is plotted in Fig. 1, right. The history-heuristic scores are only positive, and the peak is concentrated near 0, since most of the moves rarely produce a history update during the search. A second possibility for normalizing the history-heuristic scores is by dividing them by the standard deviation amongst the scores in the current position. Experimentally, we observed that the two choices to normalize the history-heuristic scores lead to similar performances. Since the second one is computationally more expensive than the first one (we have to compute the standard deviation in every position), we choose to normalize by the number of updates.

In conclusion, the score used to order the moves (*movescore*) is given by the following formula:

$$movescore = nnscore + \frac{hhscore}{hhupdate \times hhweight}$$

where *nnscore* is the score suggested by the neural network, *hhscore* is the original history-heuristic score, *hhupdate* represents the number of times the history table was updated during the search, and *hhweight* is the history weight.

To use the above formula for the move score we have to choose a value for the history weight. Choosing the proper value may have an important effect on the performance of the weighted-combination approach. If the weight is too large or too small, only one of the components of the score has influence. This effect might be beneficial in certain positions (or parts of the search tree), where one of the move-ordering methods performs poorly, but overall it is not desirable.

In essence, there are two ways to set the history weight. The first one is using off-line experiments with different values, and choosing the value leading to the best performance in these experiments. The second one adapts the weight during the search. Such an adaptive method is outlined below.

In the case of a good move ordering, the moves causing a cutoff are inspected early. Consequently, we can use the situation when the cutoff appears later as an error signal to modify the value of the history weight during the search. If the move causing a cutoff was considered later because both the neural network and the history heuristic scored it low, nothing could have been done. However, if, for instance, the neural network considered the move as promising, but it was scored overall lower than some other moves because the history heuristic scored it low, the history weight should be increased. The situation when the neural network scored a move producing a cutoff high, and the history heuristic low, we term it an *nn-miss*. The situation when the history heuristic scored the move high, but the neural network low, we term it an *hh-miss*. In the case of an nn-miss the history weight is increased by a small value β_{nn}, and in the case of an hh-miss the history weight is decreased by a small value β_{hh}. Depending on the update values, the history weight will converge to a value where the updates are balanced ($\Delta hhweight = 0$), or the ratio between the number of nn-miss (n_nnmiss) and the number of hh-miss (n_hhmiss) is inversely proportional to the ratio between the two update values:

$$n\_nnmiss \times \beta_{nn} = n\_hhmiss \times \beta_{hh}.$$

The exact values of β_{nn} and β_{hh} usually do not have a significant effect on the performance. They should have relatively small values (e.g., 0.1), in order to prevent large oscillations of the history weight, and should have similar magnitude (possibly equal) to give the same emphasis to both components.

3 Experimental Setup

In the experimental set-up we distinguish three phases: (1) the construction of the various pattern sets, (2) the training of the neural networks to predict the best move, and (3) the evaluation of the search performance of the move ordering.

In the first phase we construct the pattern sets. They consist of a board position, the set of legal moves, and the best move. To construct a pattern set we have to select the positions, to generate the legal moves in the selected positions, and to label one of the legal moves as best. The middle-game positions are selected from game databases, classified by opening. The legal moves are generated using a standard move-generation routine. A move is labeled best by one of two sources: by the choice of the player as is given in the game database, or by a game-playing program. For the latter, we employ Crafty with 2 seconds thinking time (approximately a 7-ply deep search). For the training in the second phase, we need three types of pattern sets: a *test set* to evaluate the move-ordering performance, a *validation set* for deciding when to stop the training,

Table 1. Opening lines of the learning sets.

Code	Encyclopedia Code	Number of Positions	Name of Opening Line
A30	A30	122,578	Hedgehog System of English Opening
A3x	A30-A39	685,858	Symmetrical Variation of English Opening
B84	B84	92,285	Classical Scheveningen Var. of Sicilian Defense
SI	B80-B99	1,793,305	Sicilian Defense (Scheveningen, Najdorf)
D85	D85	124,051	Exchange Variation of Grünfeld Indian
GI	D70-D99	745,911	Grünfeld Indian
E97	E97	111,536	Aronin-Taimanov Variation of King's Indian
E9x	E90-E99	865,628	Orthodox King's Indian
KI	E60-E99	2,498,964	King's Indian
ALL	A00-E99	4,412,055	all openings

and a *learning set* to modify the weights of the neural networks. For our experiments, we construct four test sets, originating from the opening lines A30, B84, D85 and E97 [11]. Each of these sets consists of 3000 positions. To each test set corresponds a validation set taken from the database with the same opening line as the test set. These sets also consists of 3000 positions. The positions from the test sets and the validation sets are different. The learning sets are taken from the same opening line as the corresponding test set or from a more general one (see below). Each learning set, except ALL, includes all available middle-game positions from its opening line excluding the positions in the test and validation sets. The learning set ALL includes 10 percent of the positions from all opening lines. For instance, the test set A30 has three corresponding learning sets: A30 with 122578 positions, A3x with 685858 positions, and ALL with 4412055 positions (see Table 1). Analogously, test set E97 has four corresponding learning sets: E97, E9x, KI and ALL (see Table 1). Other details on the learning sets are given in Table 1.

During the second phase, the neural networks are trained to predict whether a certain move is the best. The learning algorithm used is RPROP [12], known for its good generalization. Like most of the supervised-learning algorithms for neural networks, RPROP also minimizes the mean square of the error (i.e., the difference between the target value and the output value). In RPROP, however, the update of the weights uses only the sign of the derivatives, and not their size. RPROP uses an individual adaptive learning rate for each weight, which makes the algorithm more robust with respect to initial learning rates.

The error measure for evaluation and for stopping purposes is the rate of not having the highest output activation value for the 'best' move. In each epoch the entire learning set is presented. The neural network updates the weights after each epoch (i.e., batch learning). The error on the validation set is tested after each update. The training is stopped if the error on the validation set did not decrease after a certain number of epochs. In this phase we observe how often the neural network predicts the best move. For a tournament program, however, it is more important how the neural network is behaving during the search.

The third phase evaluates the information gained about the performance of the neural network. In this phase we measure the size of the search tree investigated using the neural network for move ordering. For this purpose we insert the neural network in the move ordering of Crafty and collect the statistics for search with various depths. In an internal node, Crafty considers subsequently (1) the move from the transposition table, (2) the capture moves, (3) the killer moves and (4) the remaining moves sorted according to their history-heuristic scores. In the following, we refer to this move ordering as the reference move ordering. In this list we insert the predictions of the neural-network moves, using one of the three approaches described in Sect. 2.3. The performance of a neural network is measured by dividing the size of the investigated search tree by the size of the investigated search tree without neural-network move ordering. The search performance is measured using various sets of test positions. The set of positions employed in this phase is identical to the positions from the test sets used in the training phase.

4 Experimental Results

Section 4.1 deals with the choices on the construction of the pattern sets. In particular, the choices to be made are on the best move and on the positions. Section 4.2 compares the three approaches in which the neural networks are included in the search. The results of this subsection also evaluate the performance of our move-ordering technique in comparison with the existing techniques. Section 4.3 gives insight into the predictive quality of the neural networks. Although the predictive quality is not the main evaluation criterion (the size of the investigated search tree is the main criterion), it highlights some of the strengths and weaknesses of the neural networks trained for move ordering. Moreover, an example is given for illustration.

4.1 Choices on the Construction of the Pattern Sets

The open questions on the construction of the pattern sets posed in Sect. 2.2 are about the source of the best move and the selection of the positions. Below, we describe two preliminary experiments addressing the questions. Experiment 1, called Game vs. Crafty, compares two sources of the best move: by taking the move from the game database and by taking Crafty's move. Experiment 2, called Specific vs. General, compares more opening-specific learning sets (e.g., E97) with more general learning sets (e.g., E9x, KI or ALL).

The different choices lead to different neural networks. As a measure to compare the different neural networks we take the search performance (i.e., the size of the search tree investigated using the neural networks for move ordering). Out of the three approaches to include neural networks in the search (described in Sect. 2.3), we only use the pure neural-network approach and the straightforward-combination approach. The weighted-combination approach needs additional computation to obtain the history weight, and therefore we

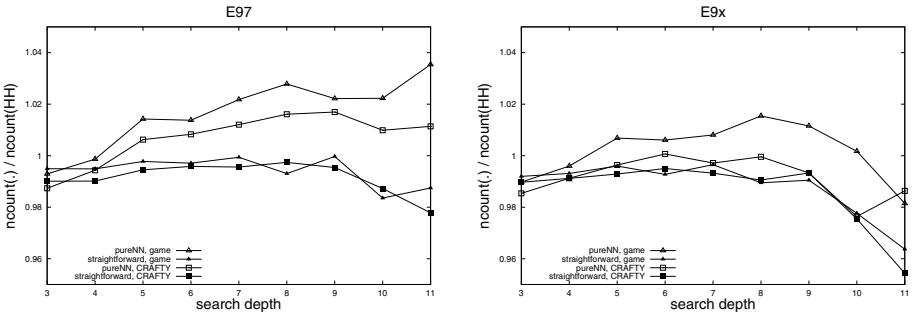

Fig. 2. Training the neural networks on game moves or on CRAFTY moves.

leave it out for the two preliminary experiments. In Figs. 2 and 3, the performance is plotted as a function of the search depth. The performance is defined as $ncount(.)/ncount(HH)$—the size of the search tree when the neural network is included divided by the size of the search tree investigated with the reference move ordering.

Game vs. CRAFTY: The first experiment is on the source of the best moves. It gives insight into the choice between the game database move and the move suggested by CRAFTY as training target. For this purpose we use the test set E97 with two learning sets: E97 and E9x. First we focus on the learning set E97. We label the moves of E97 either by the moves from the game database or by CRAFTY's suggestions. Doing so we obtain two learning sets, and accordingly, after the training process, two neural networks (one for each learning set). The search performances corresponding to the two neural networks are plotted in Fig. 2, left. The graph shows the search performance as a function of the search depth for the neural network trained with the learning set labeled either by CRAFTY or by the game database. Since we are using either the pure neural-network approach or the straightforward-combination approach, we have four curves. Similarly, Fig. 2 (right) shows the search performances corresponding to the learning set E9x. We observe that if the pure neural-network approach is used, the CRAFTY labeling is beneficial for both E97 and E9x, since it investigates a smaller search tree. If the straightforward-combination approach is used the advantage disappears. Since the CRAFTY labeling process is very time consuming[2] (especially for larger learning sets), we conclude that it is not useful to label the positions with a game program, and it is preferable to use the moves from the game database. The following experiments are using only the moves resulting from the original games (as given in the game database).

Specific vs. General: The second experiment is designed for giving an answer on how specific the learning set should be. Next to the two neural networks resulting from E97 and E9x, we trained two more neural networks with learning

[2] E.g., to label E9x we needed 20 days.

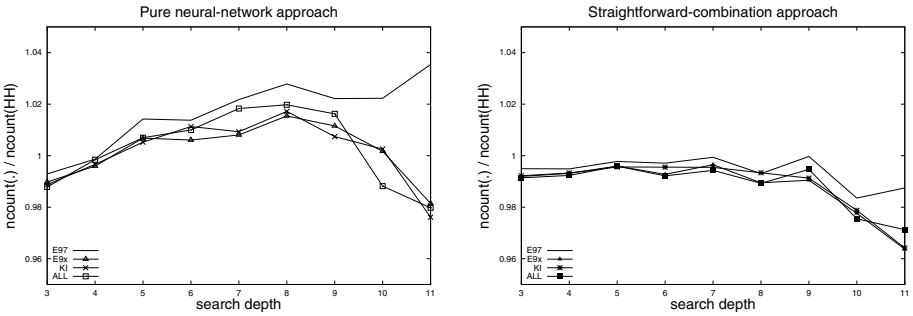

Fig. 3. Training sets varying from specific to general.

sets KI and ALL (using again E97 as test set). Out of the four sets the most specific learning set is E97, and then in this order E9x, KI, and ALL. The search performances for the four neural networks are plotted in Fig. 3. The performance of the neural networks corresponding to the learning sets E9x, KI and ALL are roughly equal. The neural network corresponding to the most specific learning set (E97) is performing worse than the other three for both the pure neural-network approach (Fig. 3, left) and the straightforward-combination approach (Fig. 3, right). The results suggest that using more specific learning sets does not improve the performance. For very specific sets, the performance may be decreased due to the lack of sufficient training data (in the case of E97: 111,536).

4.2 Comparison of the Three Neural-Network Approaches

In this subsection we compare experimentally the three approaches to include the neural network in the search. As a reference value, we also compare these approaches to the reference move ordering. To gain an even better picture on the performance of the new move-ordering techniques we use three more test sets (A30, B84 and D85), next to E97. For each of the four test sets (A30, B84, D85 and E97) a specific (A3x, SI, GI and E9x) and a general (ALL) learning set is used. With these learning sets, we obtain four neural networks specific to a certain test set and a neural network common to all test sets. The search performance is measured in the same way as in Sect. 4.1.

According to the pure neural-network approach and the straightforward-combination approach the neural networks can be included in the search directly. Thus the corresponding performance can be measured without any preparation. However, to measure the search performances for the weighted-combination approach, we first have to determine the history weight.

As described in Sect. 2.3, the history weight can be set according to two methods, either off-line or during the search. For the first method, we have to measure the search performance corresponding to different values of the history weight. Below we illustrate this mechanism for test set E97. Figure 4 plots the performance for different history-weight values, using the neural networks trained either with E9x (the left graph in Fig. 4) or with ALL (right). The

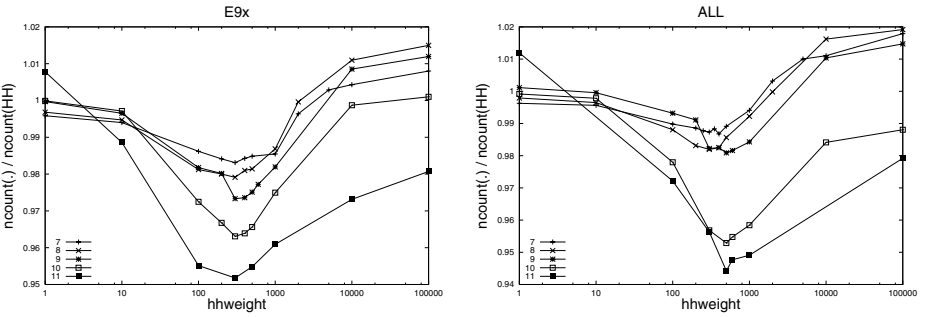

Fig. 4. Varying the history weight. The performance is plotted for the E97 test set using two neural networks, E9x and ALL, and five search depths, from 7 to 11.

five curves correspond to the performances obtained for search depths 7 to 11. The best value for the history weight is the one that corresponds to the best search performance (i.e., the lowest point for a certain curve). We observe that the best history-weight value is not changing drastically with search depth. It does, however, change with the neural network[3]. The reason for this is that the best history weight is inversely proportional to the standard deviation of the neural-network scores. As measured experimentally, the standard deviation for E9x is almost twice as large as the standard deviation for ALL, and thus the best history weight for E9x is roughly the half of the best value for ALL.

For the second method, i.e., when setting the value for the history weight during the search, using the update rules described in Sect. 2.3, we noticed that the history weight does indeed converge to a value where the equilibrium equation holds. The search performances are similar to the performances corresponding to the best static history weights from Fig. 4.

Using the value for the history weight obtained according to one of the above-mentioned methods, we measured the performance of the weighted-combination approach and compared it to the pure neural-network approach and the straightforward-combination approach. The performances of these approaches using the neural networks mentioned earlier are plotted in Fig. 5 (for test sets A30, B84, D85 and E97 separately). In the graphs, the search performance of the reference move ordering has the value of 1. From the reference move ordering, the neural-network approaches are replacing the history heuristic. Henceforth their performance is compared to that of the history heuristic. We observe, that the pure neural-network approach has a slight advantage over the history heuristic for very shallow searches (depth 3 or 4), then it has a slight disadvantage for medium depths (5 to 9), and finally again better results for depths 10 or 11. Interestingly, a similar shape (although with different values) was observed for LOA in [4][4]. The straightforward-combination approach outperforms the his-

[3] An appropriate history weight value for E9x is 300, and for ALL is 500.

[4] The reason for this particular shape is unclear to us. It might be due to some properties of the history heuristic.

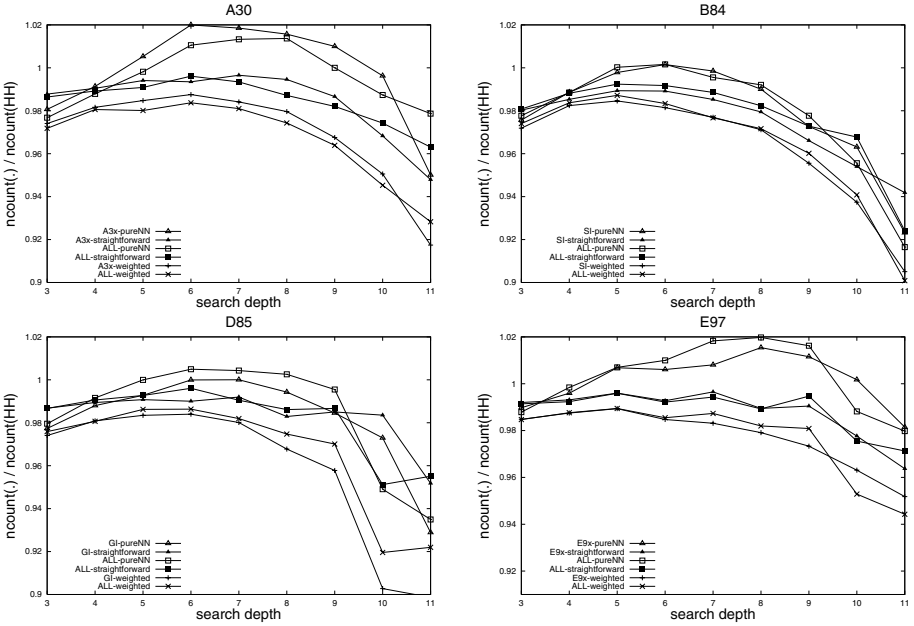

Fig. 5. Comparing the three move orderings using the four test sets (A30, B84, D85 and E97).

tory heuristic with a slight margin (approximately 1 percent) for search depths 3 to 9 and with an increasing amount for depths 10 and 11 (3 to 8 percent). The weighted-combination approach outperforms all the other ones on all test sets and search depths. The reduction compared to the history heuristic ranges between 5 and 10 percent for the largest search depth investigated.

The time overhead for the approaches using neural networks, compared to the reference move ordering, is approximately 5 to 8 percent. However, this value can be decreased by not using the neural network for move ordering close to the leaves. The performance loss in this case is negligible (less than 1 percent).

4.3 Predictive Quality of the Neural Networks

In this subsection, we provide some more insight into the predictive quality of the neural networks. In order to test the predictive quality, we measure how often the move made in the game is considered to be the best by the neural network. The neural networks used for testing are those from Sect. 4.2. The results are presented in Table 2. In the table, beside the success rate of the networks trained on specific learning sets and the ALL learning set, Crafty's predictive quality is also listed for comparison purposes. We observe that the networks are able to predict the game move in one out of three positions. This performance is almost as good as that of Crafty with a 7-ply deep search.

Table 2. Rate of correct predictions of the neural networks without any search and of CRAFTY using 7-ply searches.

test set	specific learning set	learning set ALL	CRAFTY's prediction
A30	0.33	0.32	0.39
B84	0.36	0.33	0.40
D85	0.35	0.31	0.45
E97	0.39	0.34	0.37

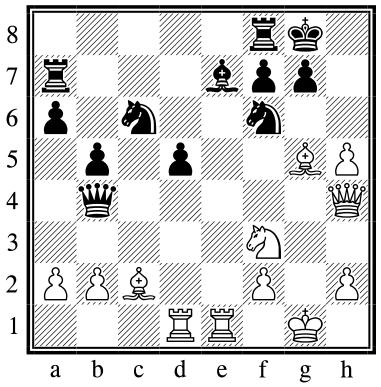

Fig. 6. A position from the Kramnik-Anand game played in Dortmund, 2001 (WTM). Kramnik's move was h6. The move ordering of the neural network: Qb4 Ne5 Bf6 a3 h6 Qh3 Bb1 Bh6 Nd4 b3 Qd4 Bb3 Rd3 Qg3 Re2 Rb1 Bc1 Bg6 Rd5 Bf5 Qf4 Re3 Rf1 Bf4 Qg4 Kh1 Kf1 Re7 h3 Bd3 Ra1 Bd2 Re5 Rc1 Re6 Bh7 Rd2 Re4 Ba4 Be3 a4 Kg2 Be4 Nd2 Qe4 Rd4 Qc4.

Below we illustrate this success by an example. Other example positions are given in the Appendix. The position in Fig. 6 is encountered between two of the best chess players, Kramnik and Anand. It is an open position, with some threats by White on the king side. The move made by Kramnik (h6), forces Black to a line of exchanges that leads in the end to an inferior ending. Without search, for the neural network the complications on the board are hidden, of course. Its main tool is to look for similarities between this position and the positions seen during the training, respectively between the legal moves in this position and the moves labeled best in the learning set. Most of the pieces in this position are in familiar places. However, the move made by White is not frequent at all in middle-game positions. Remarkably, the neural network still scores the move in the top five. Some other promising moves, like Ne5, Bf6 and Qh3 are also scored high.

5 Conclusions

This paper investigates the use of a new move-ordering heuristic, the Neural MoveMap heuristic, in chess. The heuristic uses a neural network to estimate the

likelihood of a move being the best in a certain position. The moves considered more likely to be the best are examined first. The move ordering was tested for middle-game positions using the chess program Crafty. We explored the various details of the NMM heuristic, developing a new approach to include the neural network in the search, the weighted-combination approach.

Construction of the Pattern Sets: The preliminary experiments dealt with the details of the construction of the pattern sets. From the experiments we conclude that labeling with Crafty is not giving advantage over the choice of using the moves from the game database. This is especially true if a more advanced approach is used to include the neural network in the search. Because of the extra time required by the labeling process, using the game moves is to be preferred. A second conclusion is that, although it seems counter-intuitive, using learning sets specialized on opening lines is not improving the performance. Since training one neural network instead of many (one for each opening line) is faster, and using one neural network during the search is also more convenient than to use many, we conclude that using a single general neural network is the preferable solution. In summary, we have to train one neural network with a learning set resulting directly from the game databases. Thus, the training phase can be done in a relatively short time.

Search Performance: In the main experiments we investigated the approaches to use the neural network during the search. We designed three approaches: the pure neural-network approach, the straightforward-combination approach, and the weighted-combination approach. These approaches were also compared to the reference move ordering of Crafty. Although for shallower search depths the pure neural-network approach investigated slightly larger trees, for deeper searches it proved to be superior to the reference move ordering. The two approaches that combine the neural network and the history heuristic improve upon the reference move ordering for all investigated search depths. The node reduction increases with deeper searches. Out of the two combinations, the weighted-combination approach leads to better results.

If we compare the performance of the Neural MoveMap heuristic in chess to that in LOA, we conclude that the heuristic is more beneficial in domains where there is little human knowledge on how to order the moves, but even in the presence of this knowledge, it results in a notable improvement over the existent techniques.

Both for LOA and chess, we introduced the NMM heuristic as an alternative to the history heuristic. When a game programmer decides between these two heuristics, other issues than performance have to be considered. In the following we describe the differences between the two heuristics in terms of these issues. The history heuristic is trivial to implement and requires little tuning. The NMM heuristic is more difficult to implement and requires a training phase. Neither of the two heuristics require game-dependent knowledge. In the initial development cycle speed of implementation is very attractive. In this cycle, the program is not well-tuned yet, and the NMM heuristic is likely to have a more significant impact

on the performance. Consequently, there is a trade-off between the two heuristics. In the later cycles, when the program is well developed, the performance is the most important issue for selecting the move-ordering heuristic. Therefore, the NMM heuristic should be preferred.

Predictive Quality: The predictive quality of the neural networks is surprisingly good, almost as good as a full search of Crafty. This suggests that the neural networks can be used for forward pruning too, in a similar way as in [13] for LOA. Such a forward-pruning algorithm for chess has to be developed in more detail. A promising line of research in this respect is the search control mechanism designed in [14].

References

1. Marsland, T.: A review of game-tree pruning. International Computer Chess Association Journal **9** (1986) 3–19
2. Schaeffer, J.: The history heuristic. International Computer Chess Association Journal **6** (1983) 16–19
3. Greer, K.: Computer chess move-ordering schemes using move influence. Artificial Intelligence **120** (2000) 235–250
4. Kocsis, L., Uiterwijk, J., van den Herik, J.: Move ordering using neural networks. In Monostori, L., Váncza, J., Ali, M., eds.: Engineering of Intelligent Systems. Springer-Verlag (2001) 45–50 Lecture Notes in Artificial Intelligence, Vol. 2070.
5. Tesauro, G.: Connectionist learning of expert preferences by comparison training. In Touretzky, D., ed.: Advances in Neural Information Processing Systems 1. Morgan Kaufmann (1989) 99–106
6. Utgoff, P., Clouse, J.: Two kinds of training information for evaluation function learning. In: Ninth National Conference of the American Association for Artificial Intelligence (AAAI-91), AAAI Press (1991) 596–600
7. Enderton, H.: The Golem Go program. Technical Report CMU-CS-92-101, School of Computer Science, Carnegie-Mellon University (1991)
8. Winands, M., Uiterwijk, J., van den Herik, J.: The quad heuristic in Lines of Action. International Computer Games Association Journal **24** (2001) 3–15
9. Winands, M.: Personal communication (2002)
10. Hyatt, R., Newborn, M.: Crafty goes deep. International Computer Chess Association Journal **20** (1997) 79–86
11. Matanović, A., ed.: Encyclopaedia of Chess Openings. Volume A–E. Chess Informant (2003)
12. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: IEEE International Conference on Neural Networks. (1993) 586–591
13. Kocsis, L., Uiterwijk, J., van den Herik, J.: Search-independent forward pruning. In: Belgium-Netherlands Conference on Artificial Intelligence. (2001) 159–166
14. Björnsson, Y., Marsland, T.: Learning search control in adversary games. In van den Herik, J., Monien, B., eds.: Advances in Computer Games 9. Universiteit Maastricht (2001) 157–174

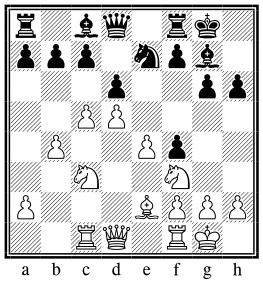

Fig. 7. An early middle-game position from the E97 test set (WTM). The 'theoretical' moves are Nd2, Nd4, cd6, a4, and h3. The move ordering of the neural network: Nd2 cd6 Nb5 Nd4 e5 Qd2 a4 h3 Re1 a3 Qd4 Nh4 Qb3 b5 Qa4 c6 h4 Nb1 Qe1 Ne5 Ne1 Na4 Qd3 Ba6 Qc2 g4 Kh1 Nd3 Bb5 Ra1 Rb1 Rc2 Bc4 g3 Ng5.

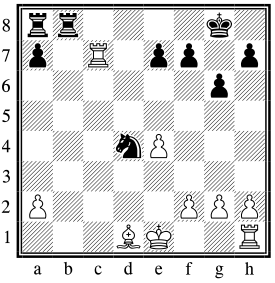

Fig. 8. A position close to the endgame from the D85 test set (BTM). The move made in the game was Kf8. The move ordering of the neural network: Rb2 Rb1 Nb5 Rf8 Ne6 Rb3 Rb6 a5 Nc2 e6 Re8 Kg7 Rd8 g5 Ne2 e5 f5 Rb4 h6 f6 Nc6 h5 Kh8 a6 Nb3 Kf8 Nf3 Nf5 Rb5 Rc8 Rb7.

Appendix: Examples of Move Ordering Using the Neural Network

In Figs. 7, 8 and 9, we illustrate some of the weaknesses and strengths of the move ordering performed with the neural networks. The positions in the figures were not included in the learning sets. The neural network employed to order the moves was the one trained with positions resulting from all opening lines (ALL). The first two positions are from the edges of the middle-game phase, illustrating some interesting properties of these parts of the game. The third position is a closed position from the 'core' of the middle game.

In the position in Fig. 7, an early middle-game position, the moves typical for this line of opening are easy to recognize. Consequently, although the position was never 'seen' by the neural network, the move ordering suggested is excellent, rating the 5 'theoretical' moves among the top 8 moves.

The second position (Fig. 8) is close to the endgame. Although, some of the first-rated moves make some sense, the good moves (including the one made in the game), are in the tail of the list. The shortcoming is resulting from the fact

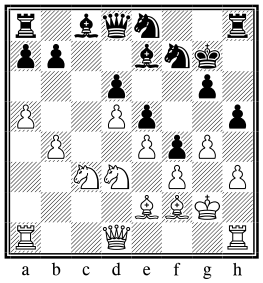

Fig. 9. A position from a tournament game played by the first author (WTM). White's move Qg1 forces black to play a6, allowing Bb6. The move ordering of the neural network: h4 Bb6 Qg1 Qb3 Re1 Nf4 Nb5 a6 Qd2 gh5 g5 Ne5 Bf1 Ba7 Bg3 Rc1 Kh2 Qc1 Ra3 Bh4 Qb1 Rf1 Na4 Rg1 Kf1 Rh2 Ra2 Rb1 Qf1 Ra4 Bd4 Be1 Qe1 Qc2 Bg1 Na2 Ne1 Nb1 Nc5 Qa4 Be3 Nc1 Bc5 b5 Kg1 Nb2.

that in the position the middle-game principles are not true anymore, and it is completely natural to move the king closer to the center of the board. This suggests that if too many simplifications were made, and especially if the queens disappeared from the board, it is better not to use the neural networks for move ordering, even if under some criteria the position is considered to be the middle game.

The third position (Fig. 9) is a closed middle-game position (as opposed to the open position of Fig. 6). Although the move made in the game is again atypical, the neural network scores it relatively high again.